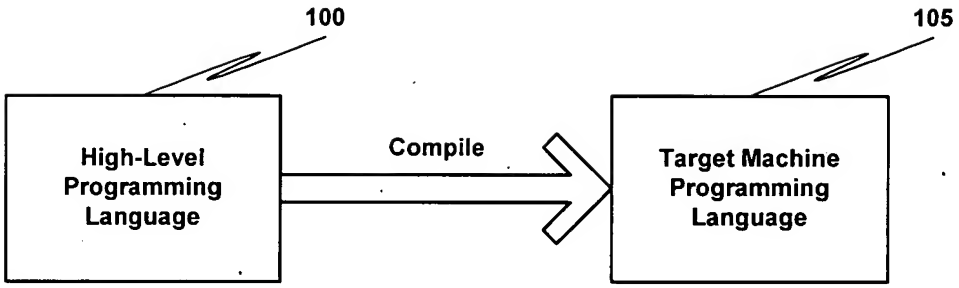
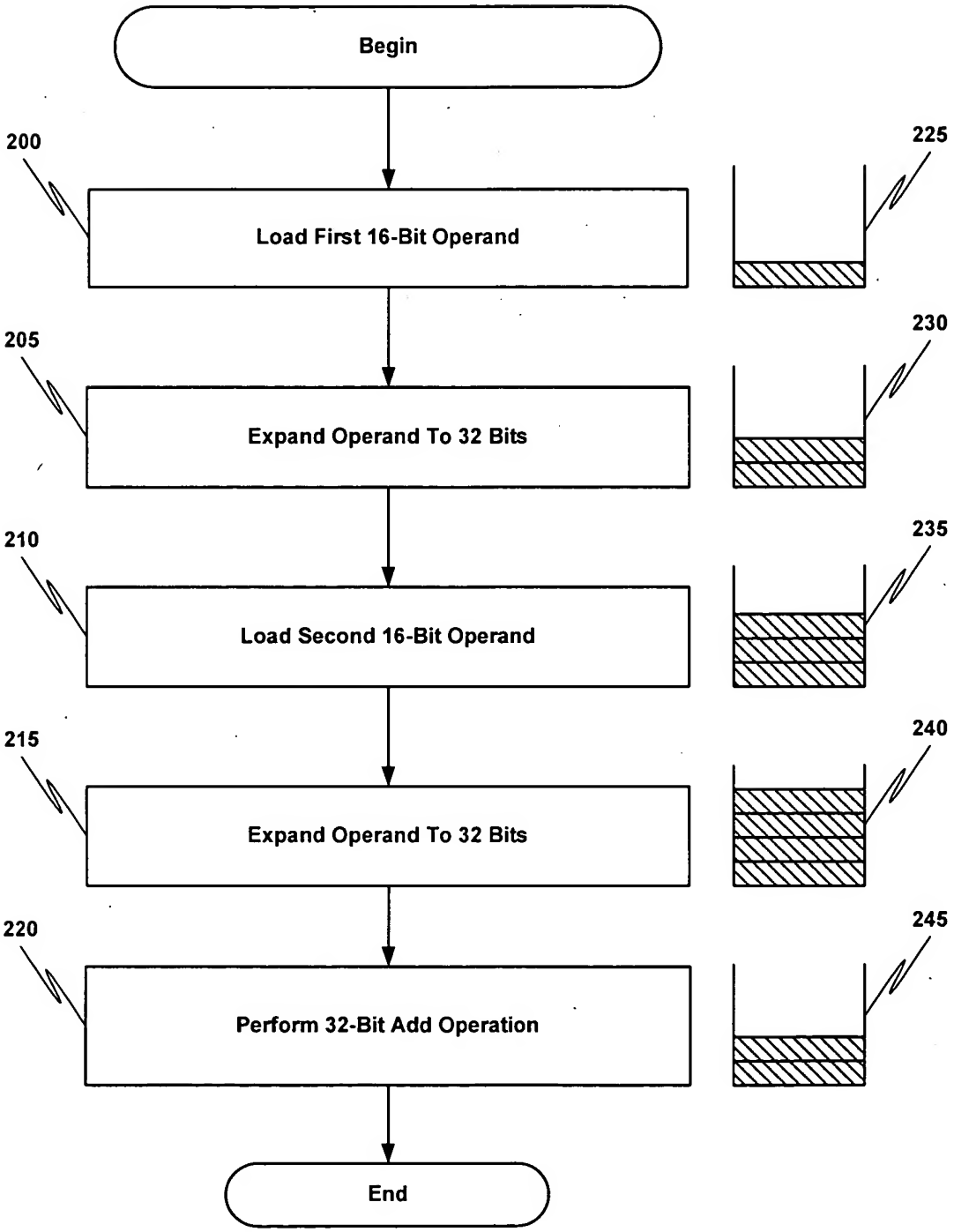


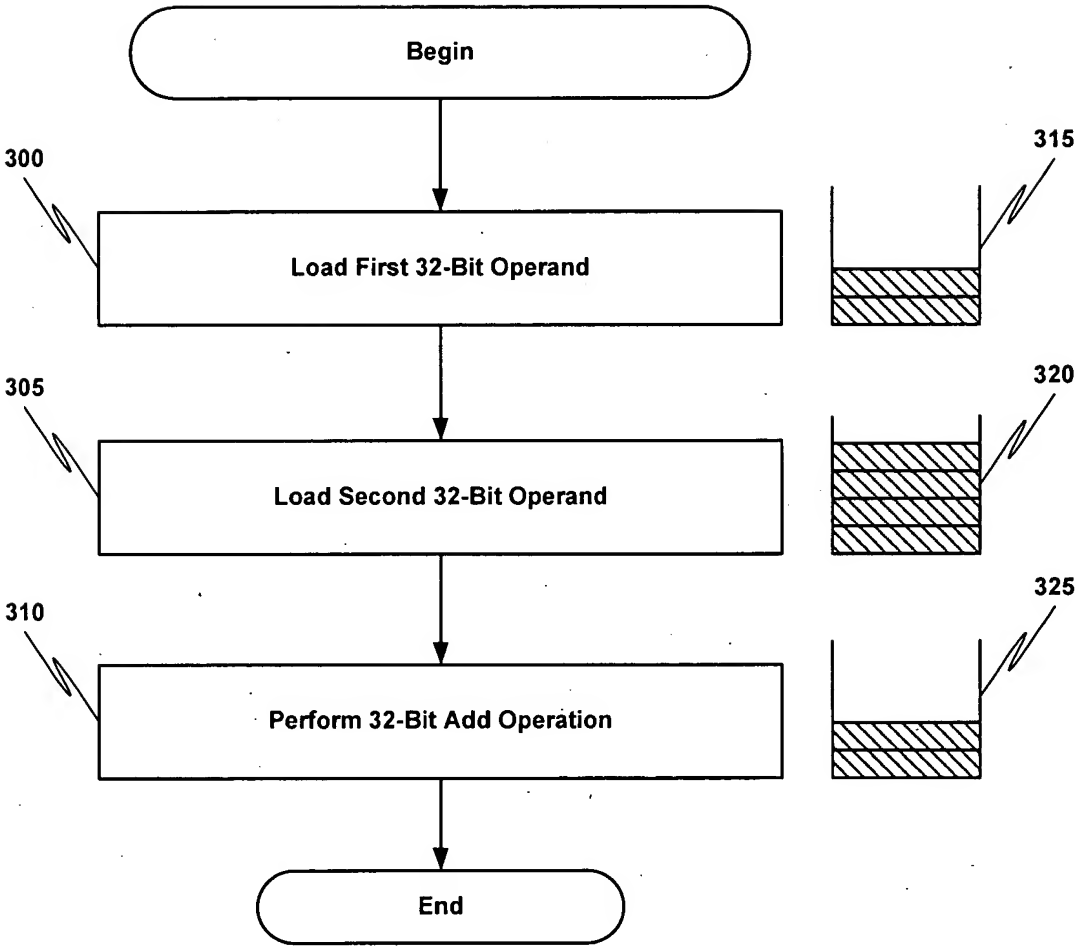
Prior Art

FIG. 1



Prior Art

FIG. 2



Prior Art

FIG. 3

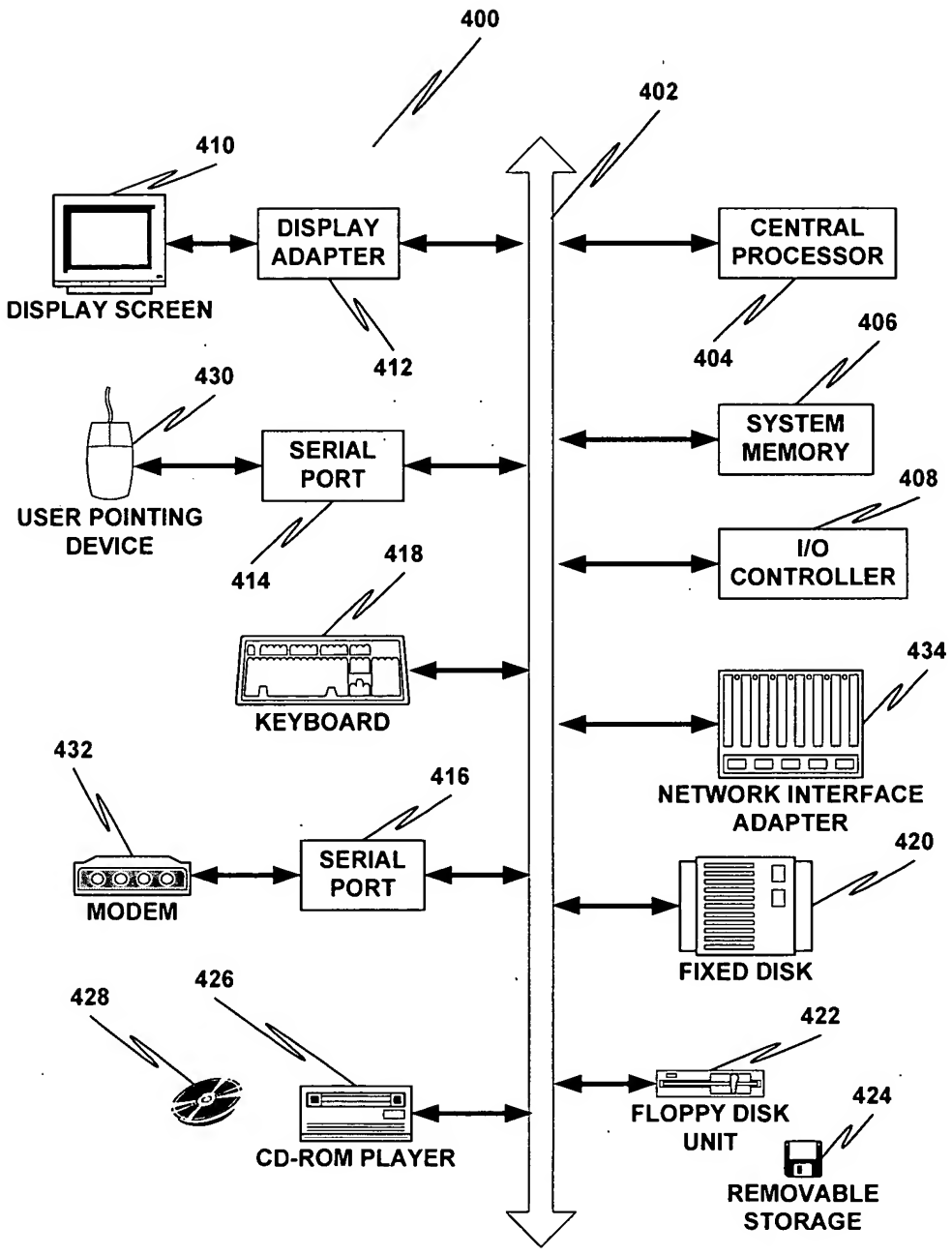

400
402
410
DISPLAY ADAPTER
CENTRAL PROCESSOR
DISPLAY SCREEN
412
404
406
430
SERIAL PORT
SYSTEM MEMORY
USER POINTING DEVICE
408
I/O CONTROLLER
414
418
KEYBOARD
434
NETWORK INTERFACE ADAPTER
432
416
SERIAL PORT
420
MODEM
FIXED DISK
428
426
422
FLOPPY DISK UNIT
CD-ROM PLAYER
424
REMOVABLE STORAGE

FIG. 4

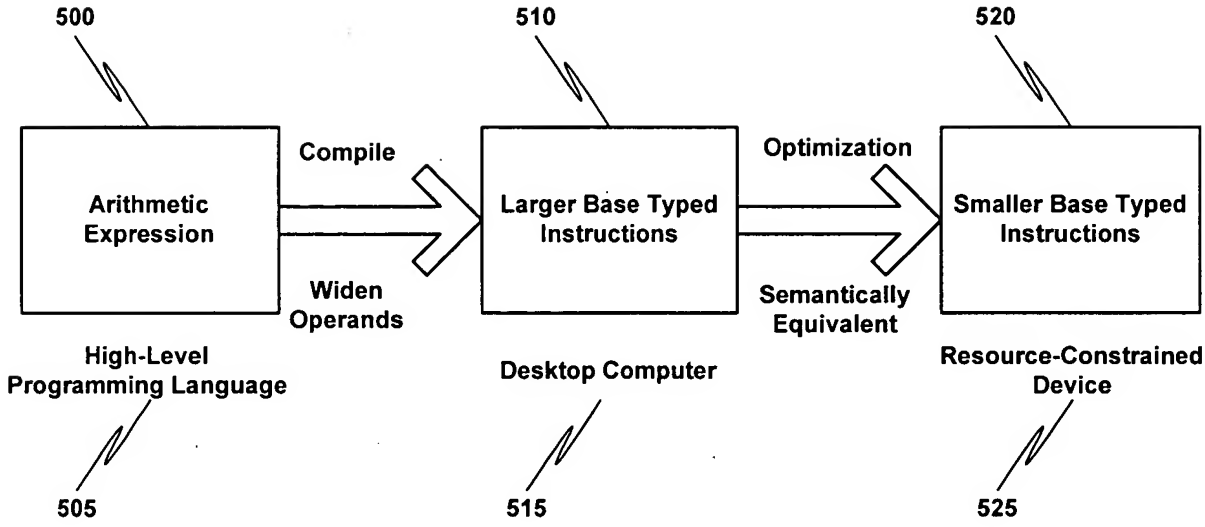

500
510
520
Arithmetic Expression
Compile
Widen Operands
Larger Base Typed Instructions
Optimization
Semantically Equivalent
Smaller Base Typed Instructions
High-Level Programming Language
Desktop Computer
Resource-Constrained Device
505
515
525

FIG. 5

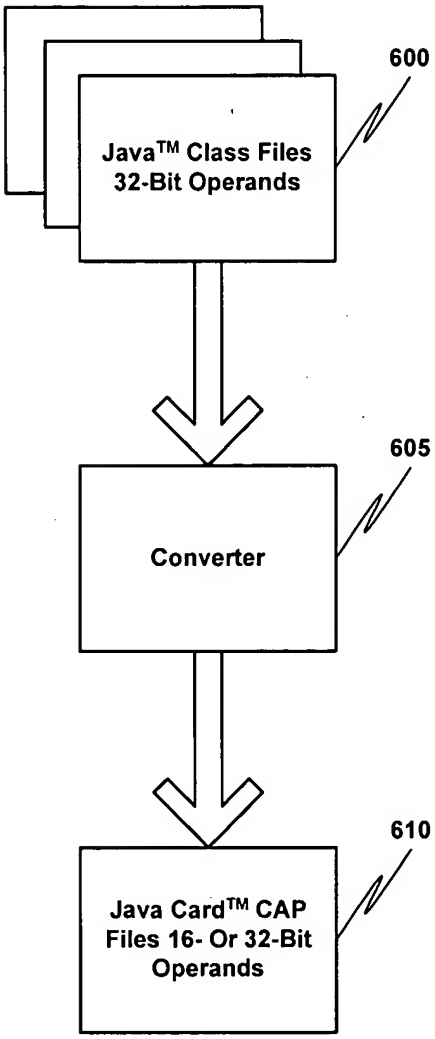

600
Java™ Class Files
32-Bit Operands
605
Converter
610
Java Card™ CAP
Files 16- Or 32-Bit
Operands

FIG. 6

Desktop Computer

```
short a = 0x7FFF;
short b = 1;
short c = 1;

result = a [int-type add] b;
       = 0x00007FFF + 1
       = 0x00008000
       = 32768
```

FIG. 7A

Resource-Constrained Device

```
short a = 0x7FFF;
short b = 1;
short c = 1;

result = a [short-type add] b;
       = 0x7FFF + 1
       = 0x8000
       = -32768
```

FIG. 7B

```
result = (short) a [int-type add] b;
       = (short) (0x00007FFF + 1)
       = (short) (0x00008000)
       = 0xFFFF800
       = -32768
```

FIG. 8A

```
result = (short) a [short-type add] b;
       = (short) (0x7FFF + 1)
       = (short) (0x8000)

       = -32768
```

FIG. 8B

```
result = (short) (a [int-type add] b
         [int-type add] c);
       = (short) (0x00007FFF + 1 + 1)
       = (short) (0x00008001)
       = 0xFFFF8001
       = -32767
```

FIG. 9A

```
result = (short) (a [short-type add] b
         [short-type add] c);
       = (short) (0x7FFF + 1 + 1)
       = (short) (0x8001)
       = -32767
```

FIG. 9B

```
result = a[int-type add] b
         [int-type divide] c;
       = (0x00007FFF + 1) / 2
       = 0x00008000 / 2
       = 0x00004000
       = 16384
```

FIG. 10A

```
result = a [short-type add] b
         [short-type divide] c;
       = (0x7FFF + 1) / 2
       = 0x8000 / 2
       = 0xC000
       = -16384
```

FIG. 10B

FIG. 7A-10B

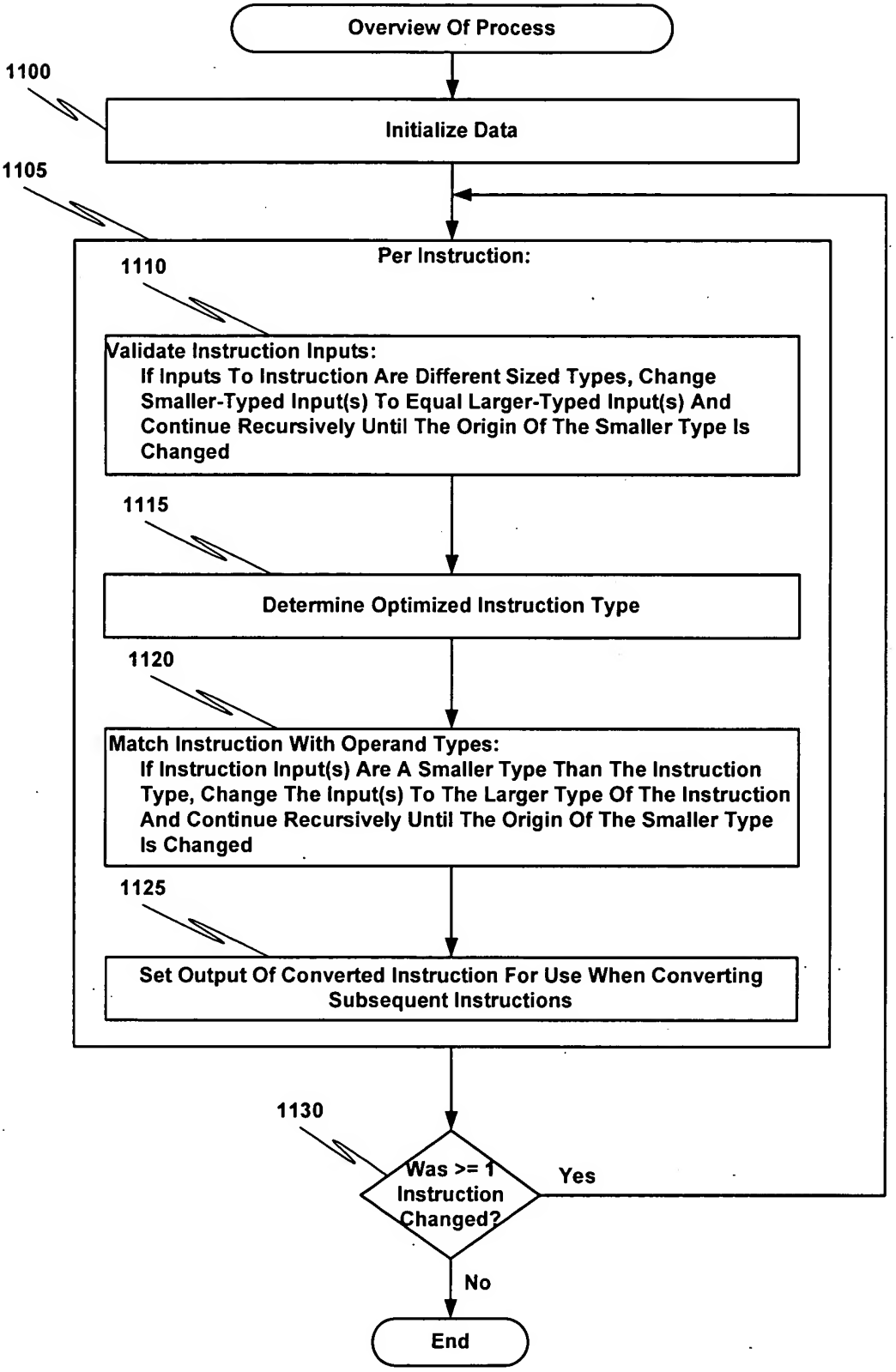

Overview Of Process
1100
Initialize Data
1105
Per Instruction:
1110
Validate Instruction Inputs:
If Inputs To Instruction Are Different Sized Types, Change Smaller-Typed Input(s) To Equal Larger-Typed Input(s) And Continue Recursively Until The Origin Of The Smaller Type Is Changed
1115
Determine Optimized Instruction Type
1120
Match Instruction With Operand Types:
If Instruction Input(s) Are A Smaller Type Than The Instruction Type, Change The Input(s) To The Larger Type Of The Instruction And Continue Recursively Until The Origin Of The Smaller Type Is Changed
1125
Set Output Of Converted Instruction For Use When Converting Subsequent Instructions
1130
Was >= 1 Instruction Changed?
Yes
No
End

FIG. 11

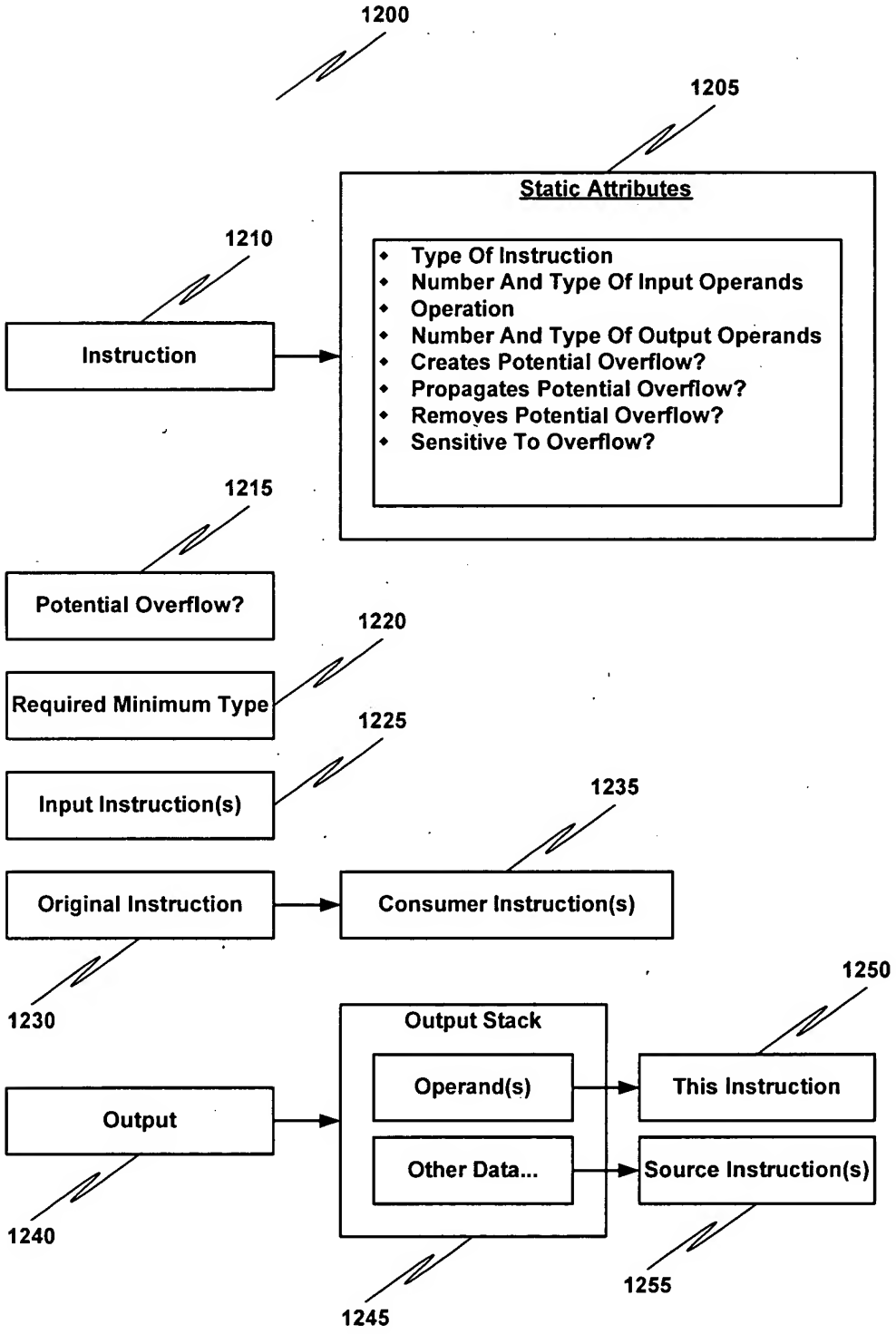

1200
1205
Static Attributes
Type Of Instruction
Number And Type Of Input Operands
Operation
Number And Type Of Output Operands
Creates Potential Overflow?
Propagates Potential Overflow?
Removes Potential Overflow?
Sensitive To Overflow?
1210
Instruction
1215
Potential Overflow?
1220
Required Minimum Type
1225
Input Instruction(s)
1235
Original Instruction
Consumer Instruction(s)
1230
1250
Output Stack
Operand(s)
This Instruction
Output
Other Data...
Source Instruction(s)
1240
1245
1255

FIG. 12

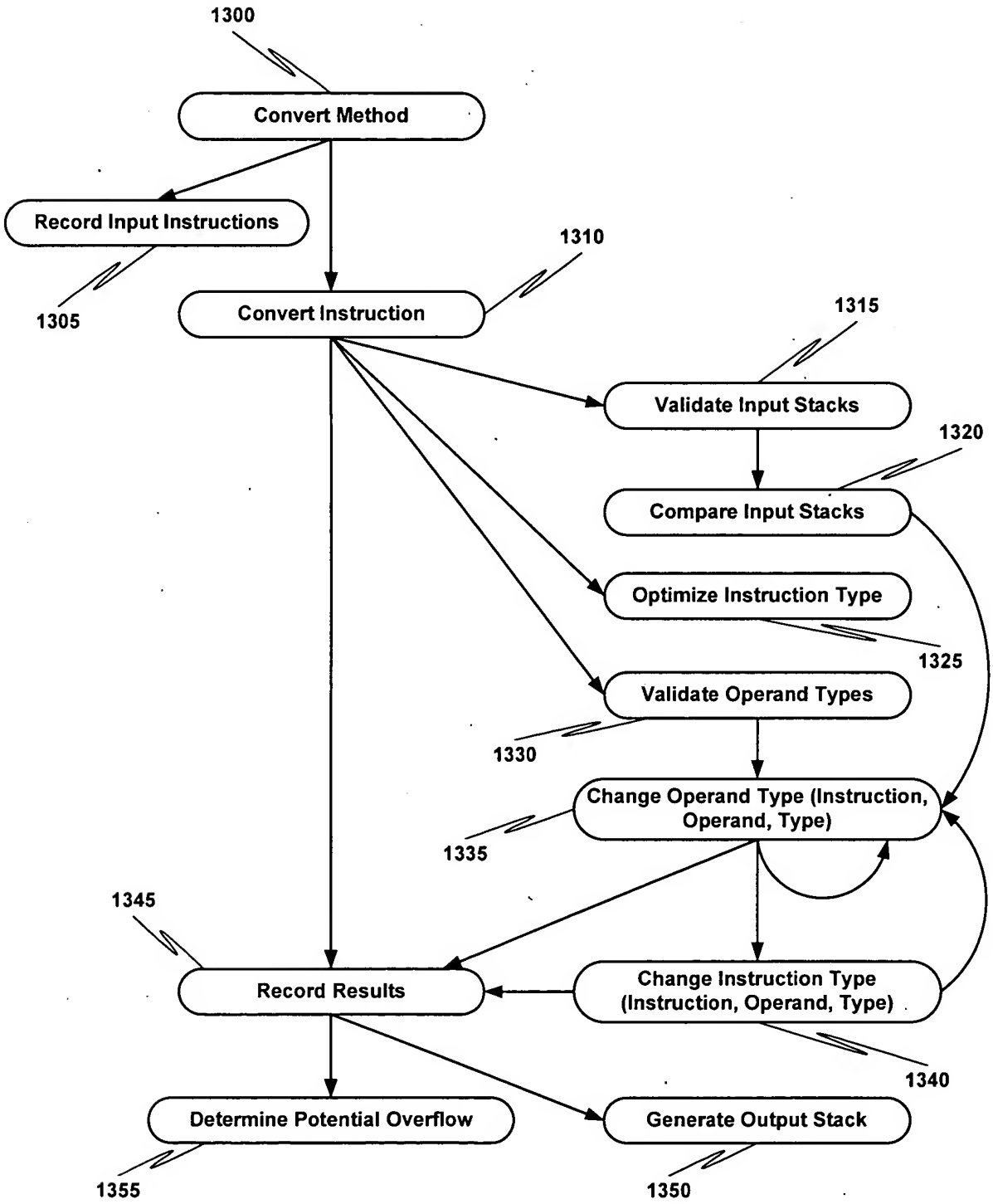

1300
Convert Method
Record Input Instructions
1305
1310
Convert Instruction
1315
Validate Input Stacks
1320
Compare Input Stacks
Optimize Instruction Type
1325
Validate Operand Types
1330
Change Operand Type (Instruction, Operand, Type)
1335
1345
Record Results
Change Instruction Type (Instruction, Operand, Type)
1340
Determine Potential Overflow
1355
Generate Output Stack
1350

FIG. 13

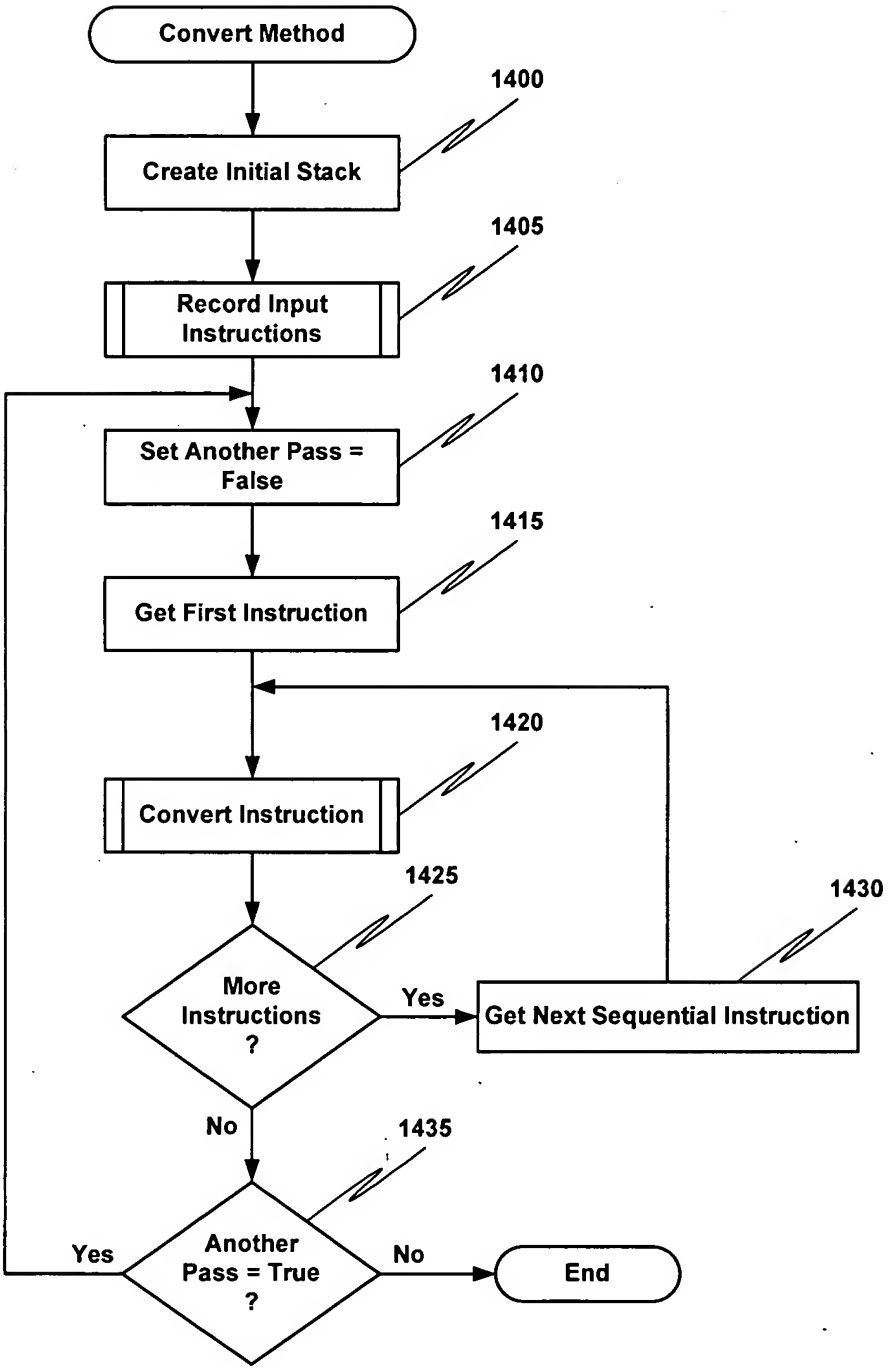

Convert Method
1400
Create Initial Stack
1405
Record Input Instructions
1410
Set Another Pass = False
1415
Get First Instruction
1420
Convert Instruction
1425
More Instructions ?
Yes
1430
Get Next Sequential Instruction
No
1435
Another Pass = True ?
Yes
No
End

FIG. 14

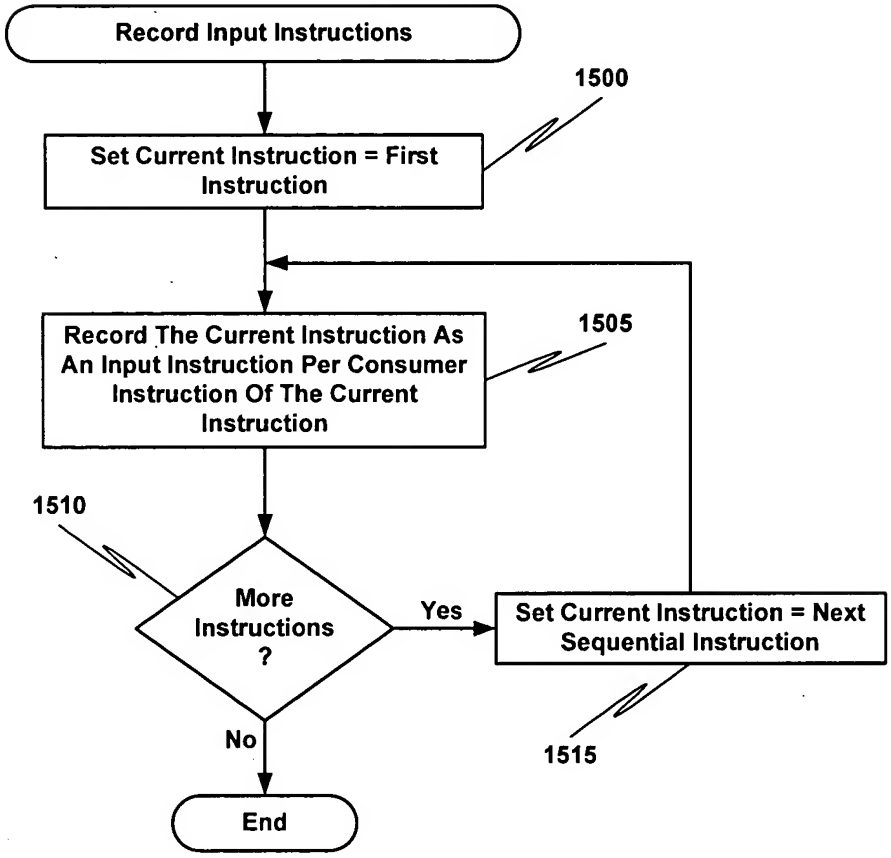

Record Input Instructions
1500
Set Current Instruction = First Instruction
Record The Current Instruction As An Input Instruction Per Consumer Instruction Of The Current Instruction
1505
1510
More Instructions ?
Yes
Set Current Instruction = Next Sequential Instruction
1515
No
End

FIG. 15

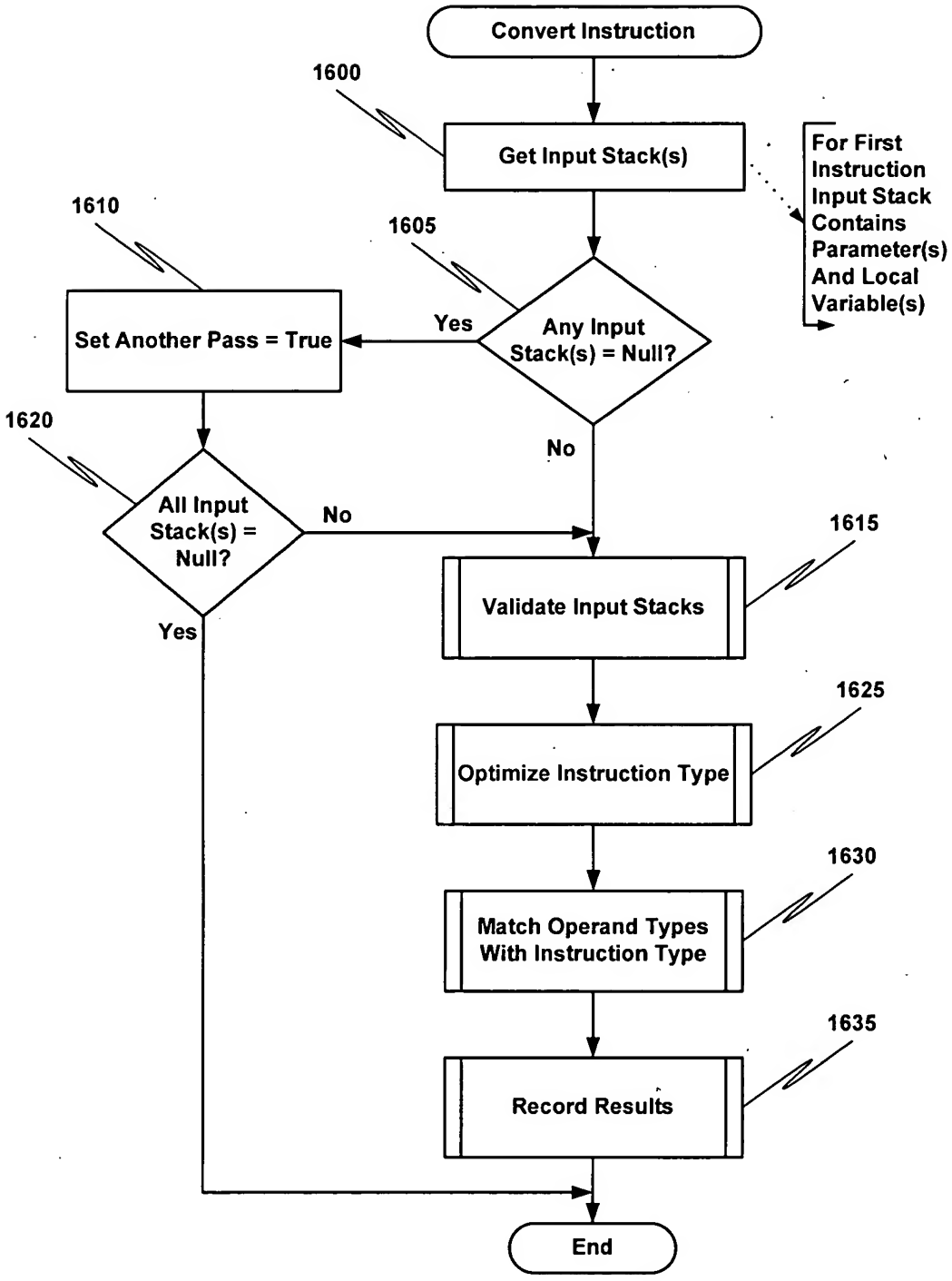

Convert Instruction
1600
Get Input Stack(s)
For First Instruction Input Stack Contains Parameter(s) And Local Variable(s)
1610
1605
Set Another Pass = True
Yes
Any Input Stack(s) = Null?
1620
No
All Input Stack(s) = Null?
No
1615
Validate Input Stacks
Yes
1625
Optimize Instruction Type
1630
Match Operand Types With Instruction Type
1635
Record Results
End

FIG. 16

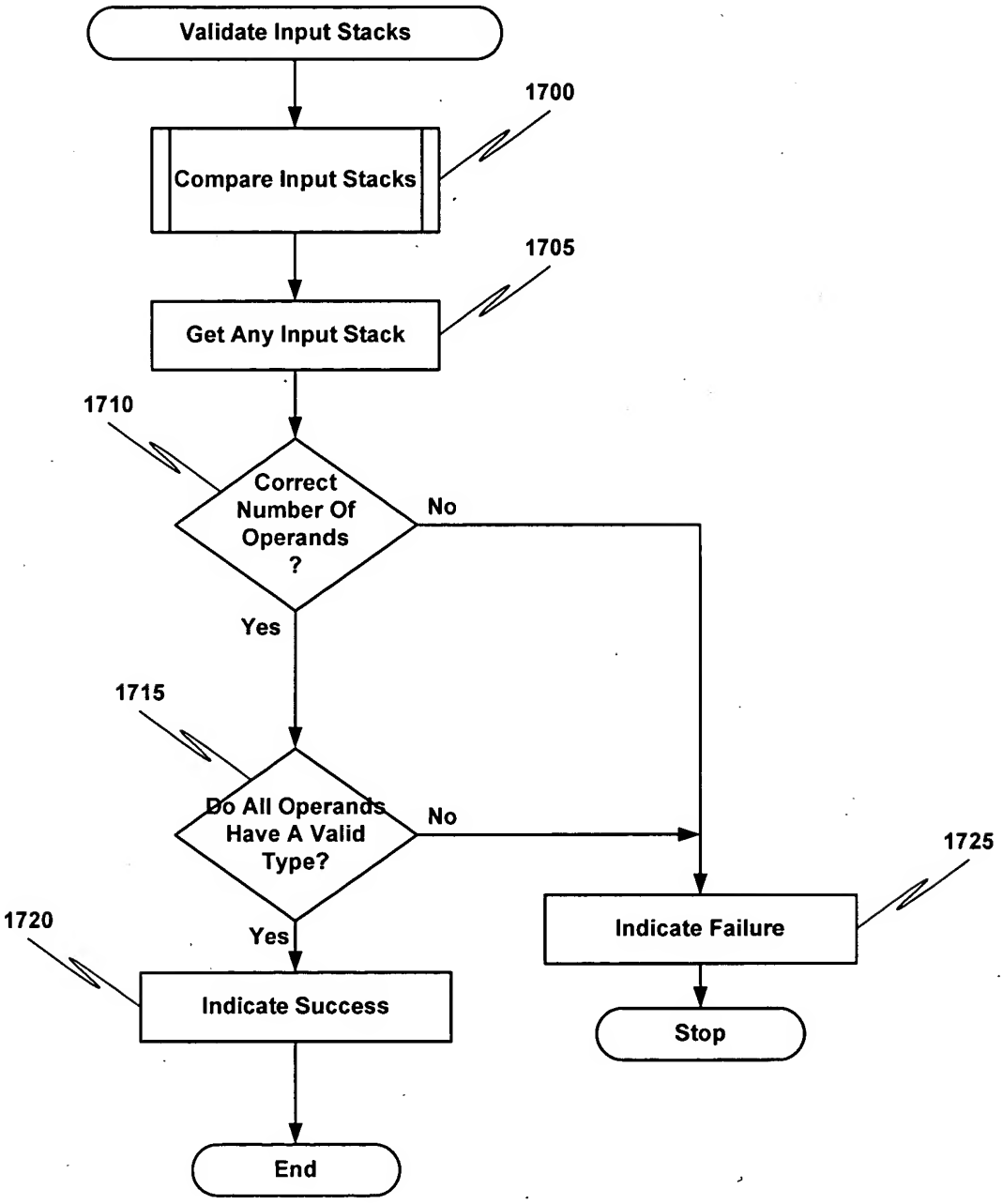

Validate Input Stacks
1700
Compare Input Stacks
1705
Get Any Input Stack
1710
Correct Number Of Operands ?
No
Yes
1715
Do All Operands Have A Valid Type?
No
1725
Indicate Failure
Yes
1720
Indicate Success
Stop
End

FIG. 17

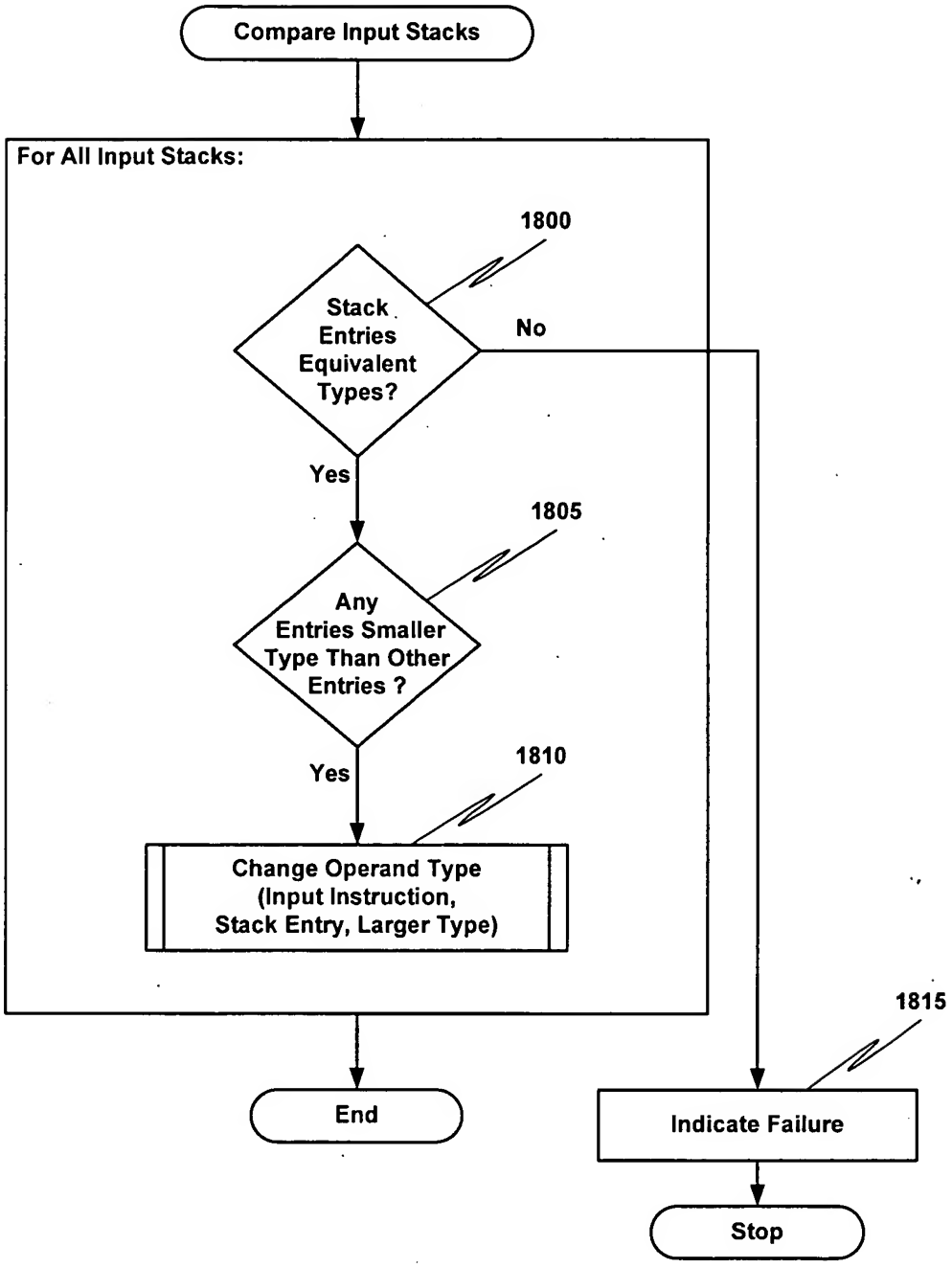

Compare Input Stacks
For All Input Stacks:
1800
Stack Entries Equivalent Types?
No
Yes
1805
Any Entries Smaller Type Than Other Entries ?
Yes
1810
Change Operand Type (Input Instruction, Stack Entry, Larger Type)
1815
End
Indicate Failure
Stop

FIG. 18

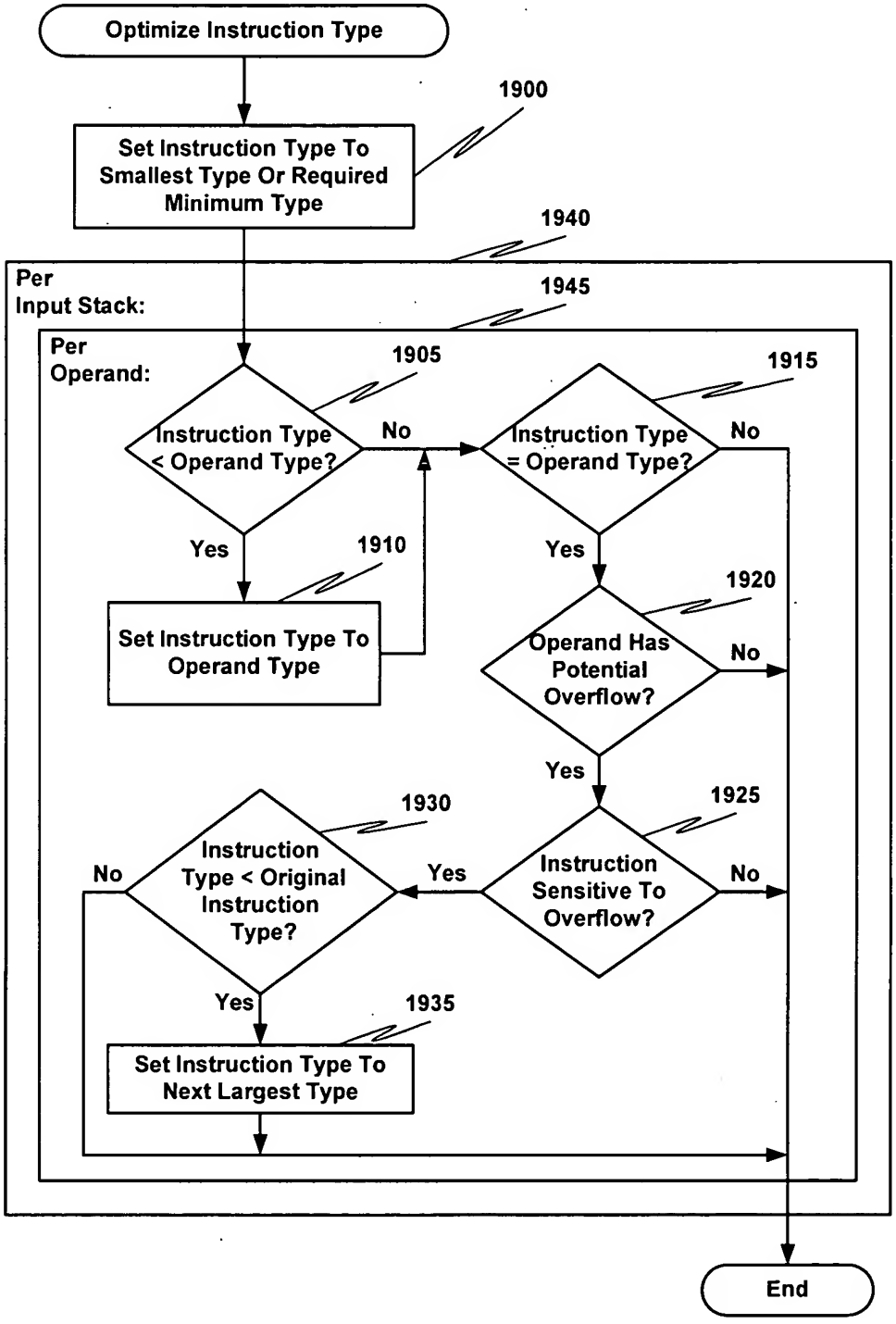

Optimize Instruction Type
1900
Set Instruction Type To Smallest Type Or Required Minimum Type
1940
Per Input Stack:
1945
Per Operand:
1905
Instruction Type < Operand Type?
No
1915
Instruction Type = Operand Type?
No
Yes
1910
Set Instruction Type To Operand Type
Yes
1920
Operand Has Potential Overflow?
No
Yes
1930
Instruction Type < Original Instruction Type?
No
Yes
1925
Instruction Sensitive To Overflow?
No
Yes
1935
Set Instruction Type To Next Largest Type
End

FIG. 19

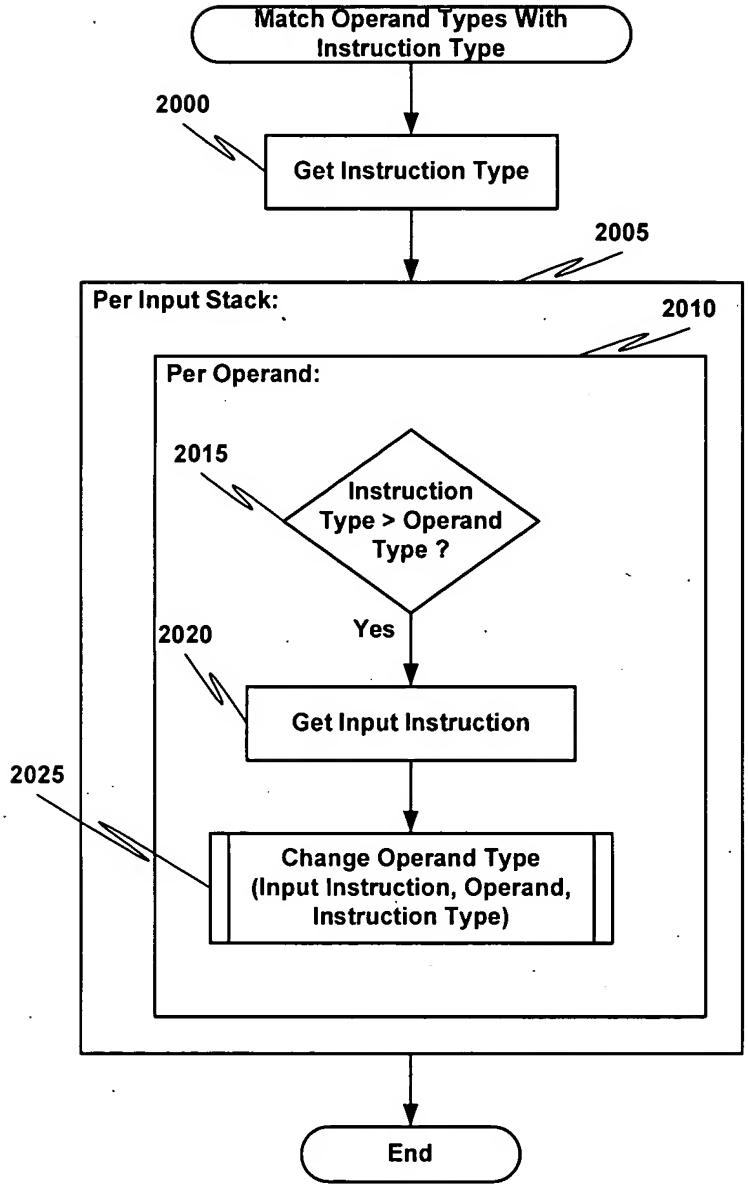

Match Operand Types With Instruction Type
2000
Get Instruction Type
2005
Per Input Stack:
2010
Per Operand:
2015
Instruction Type > Operand Type ?
Yes
2020
Get Input Instruction
2025
Change Operand Type (Input Instruction, Operand, Instruction Type)
End

FIG. 20

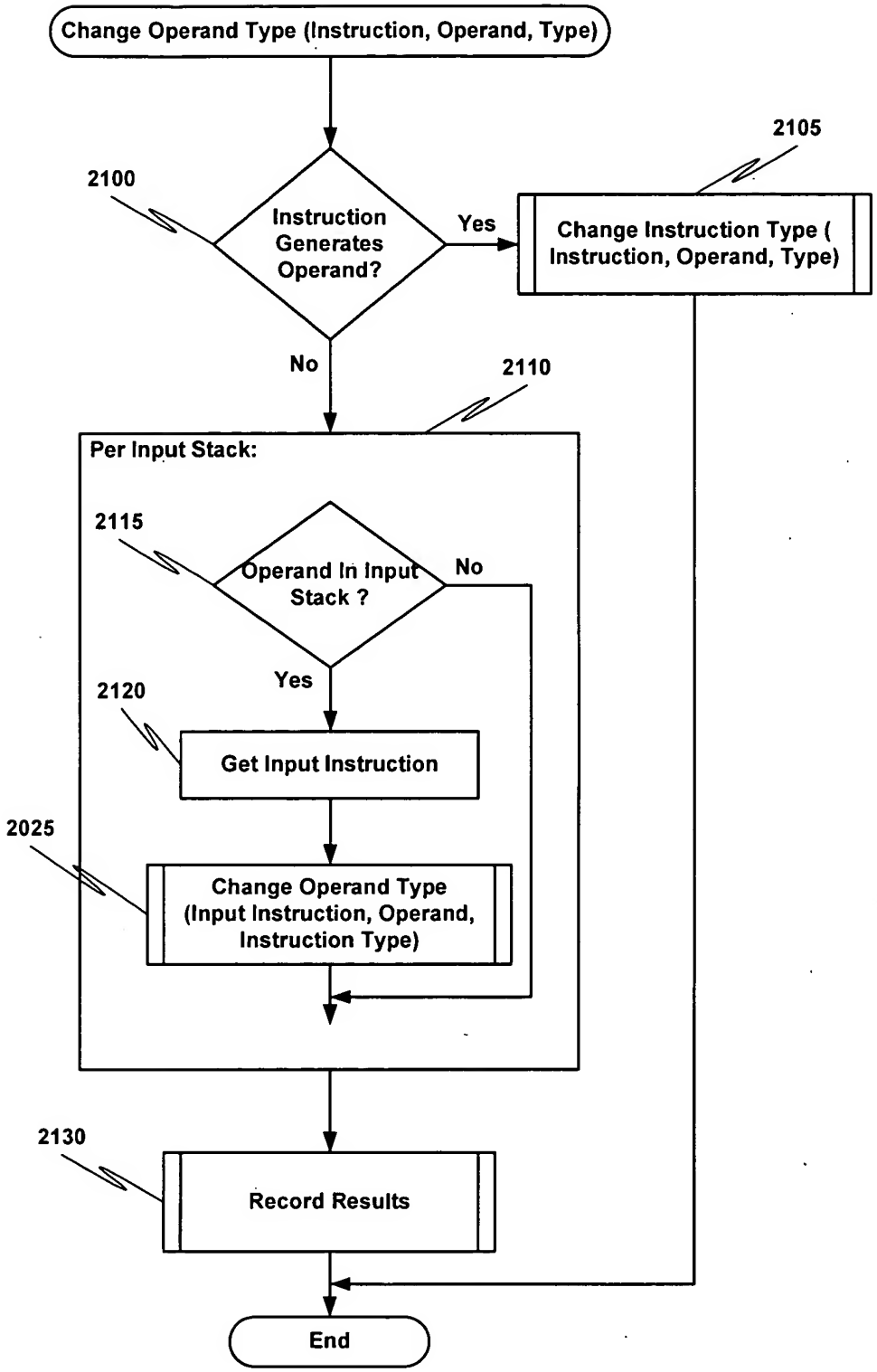

Change Operand Type (Instruction, Operand, Type)
2105
2100
Instruction Generates Operand?
Yes
Change Instruction Type (Instruction, Operand, Type)
No
2110
Per Input Stack:
2115
Operand In Input Stack ?
No
Yes
2120
Get Input Instruction
2025
Change Operand Type (Input Instruction, Operand, Instruction Type)
2130
Record Results
End

FIG. 21

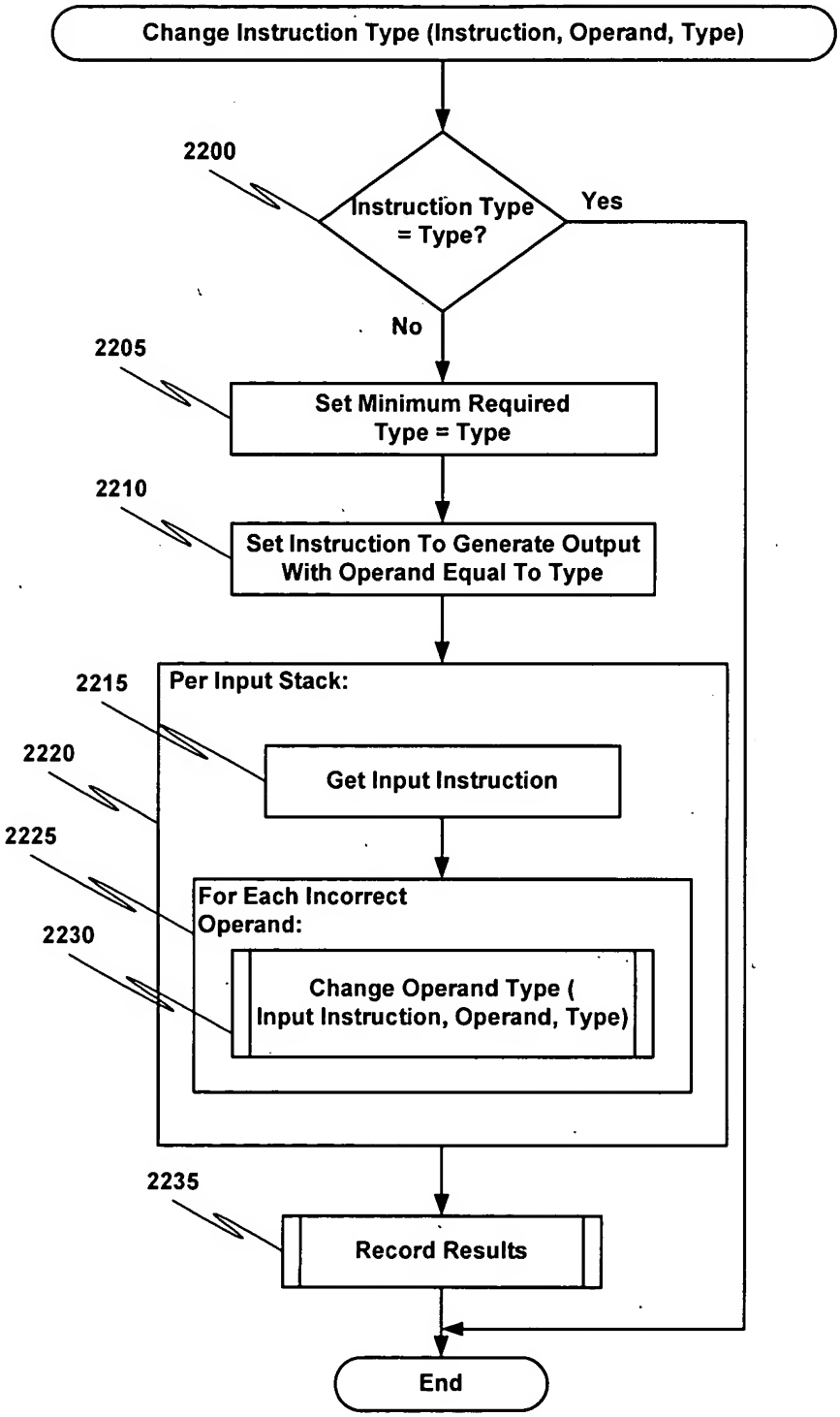

Change Instruction Type (Instruction, Operand, Type)
2200
Instruction Type = Type?
Yes
No
2205
Set Minimum Required Type = Type
2210
Set Instruction To Generate Output With Operand Equal To Type
2215
Per Input Stack:
2220
Get Input Instruction
2225
For Each Incorrect Operand:
2230
Change Operand Type (Input Instruction, Operand, Type)
2235
Record Results
End

FIG. 22

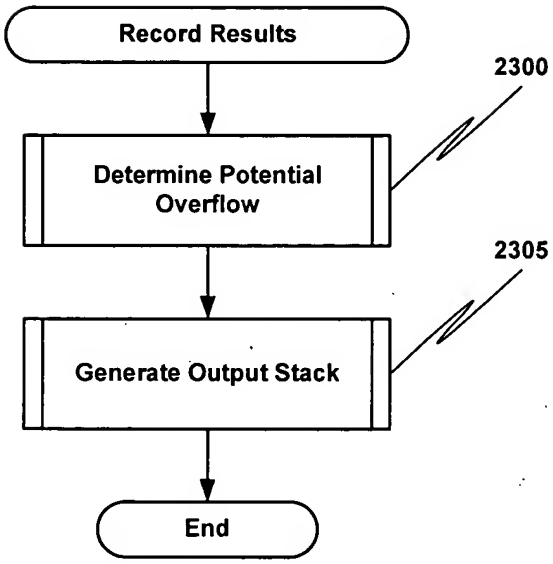

Record Results
2300
Determine Potential Overflow
2305
Generate Output Stack
End

FIG. 23

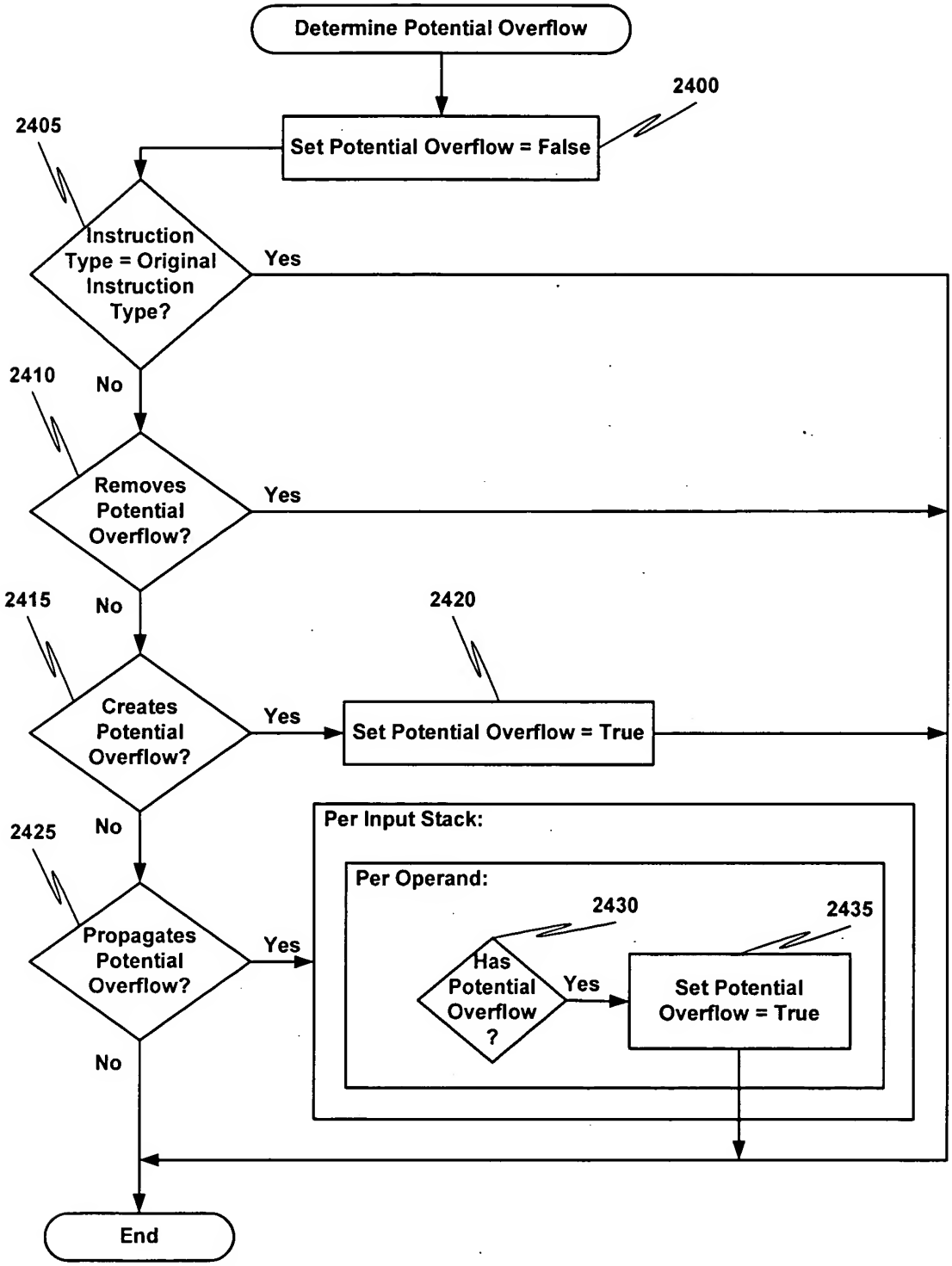

Determine Potential Overflow
2400
Set Potential Overflow = False
2405
Instruction Type = Original Instruction Type?
Yes
No
2410
Removes Potential Overflow?
Yes
No
2415
2420
Creates Potential Overflow?
Yes
Set Potential Overflow = True
No
2425
Per Input Stack:
Per Operand:
2430
2435
Propagates Potential Overflow?
Yes
Has Potential Overflow ?
Yes
Set Potential Overflow = True
No
End

FIG. 24A

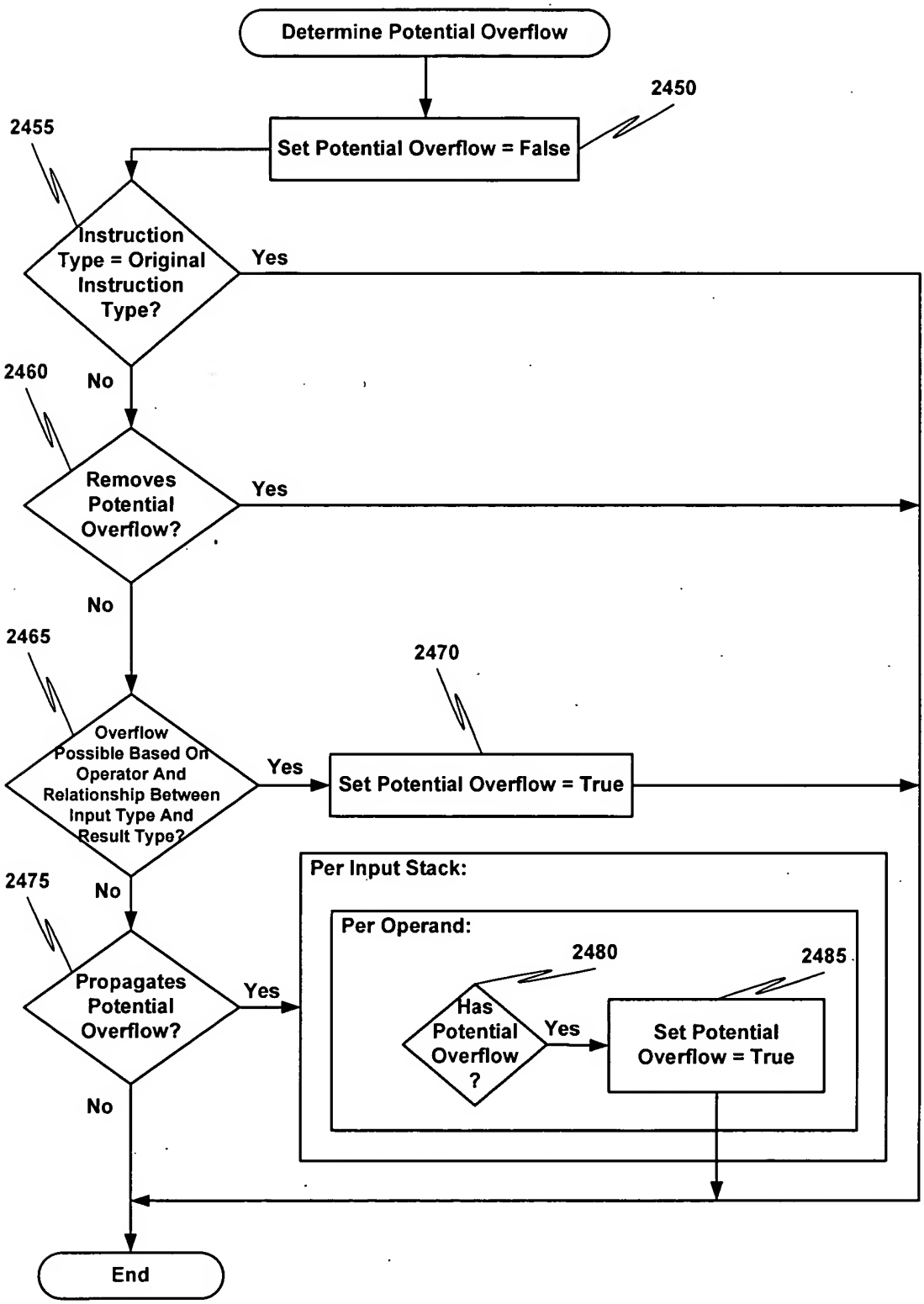

Determine Potential Overflow
2450
Set Potential Overflow = False
2455
Instruction Type = Original Instruction Type?
Yes
No
2460
Removes Potential Overflow?
Yes
No
2465
2470
Overflow Possible Based On Operator And Relationship Between Input Type And Result Type?
Yes
Set Potential Overflow = True
2475
No
Per Input Stack:
Per Operand:
2480
2485
Has Potential Overflow ?
Yes
Set Potential Overflow = True
Propagates Potential Overflow?
Yes
No
End

FIG. 24B

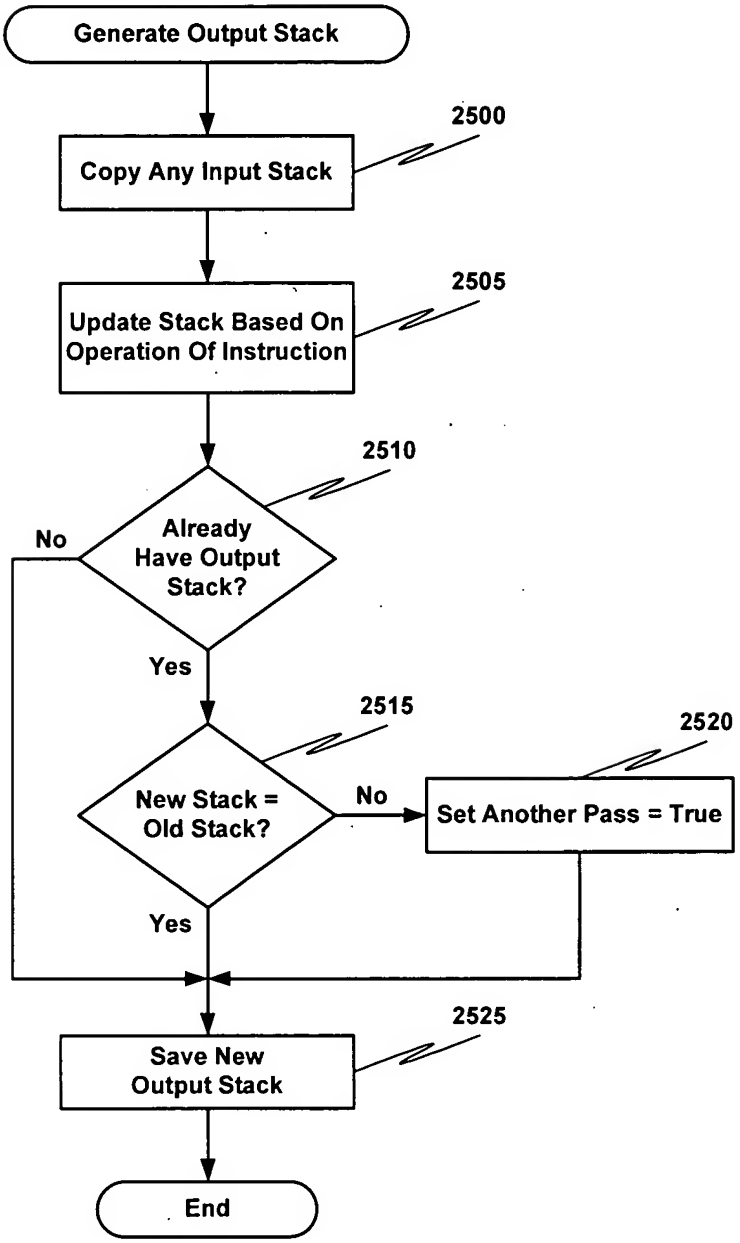

Generate Output Stack
2500
Copy Any Input Stack
2505
Update Stack Based On Operation Of Instruction
2510
Already Have Output Stack?
No
Yes
2515
New Stack = Old Stack?
No
2520
Set Another Pass = True
Yes
2525
Save New Output Stack
End

FIG. 25

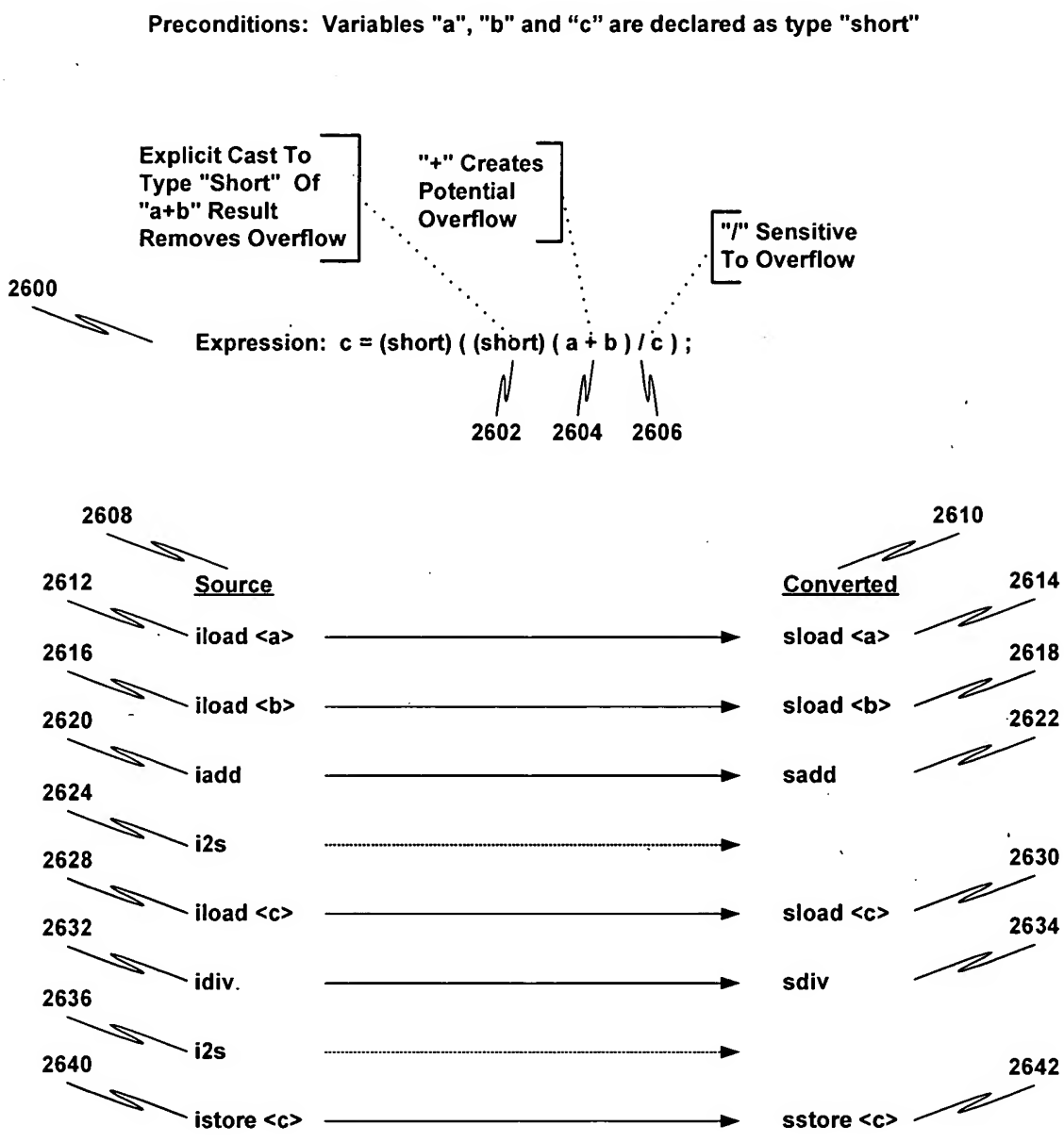

Preconditions: Variables "a", "b" and "c" are declared as type "short"
Explicit Cast To Type "Short" Of "a+b" Result Removes Overflow
"+" Creates Potential Overflow
"/" Sensitive To Overflow
2600
Expression: c = (short) ((short) (a + b) / c) ;
2602
2604
2606
2608
2610
2612
Source
Converted
2614
iload <a>
sload <a>
2616
2618
iload <b>
sload <b>
2620
2622
iadd
sadd
2624
i2s
2628
2630
iload <c>
sload <c>
2632
2634
idiv.
sdiv
2636
i2s
2640
2642
istore <c>
sstore <c>

FIG. 26

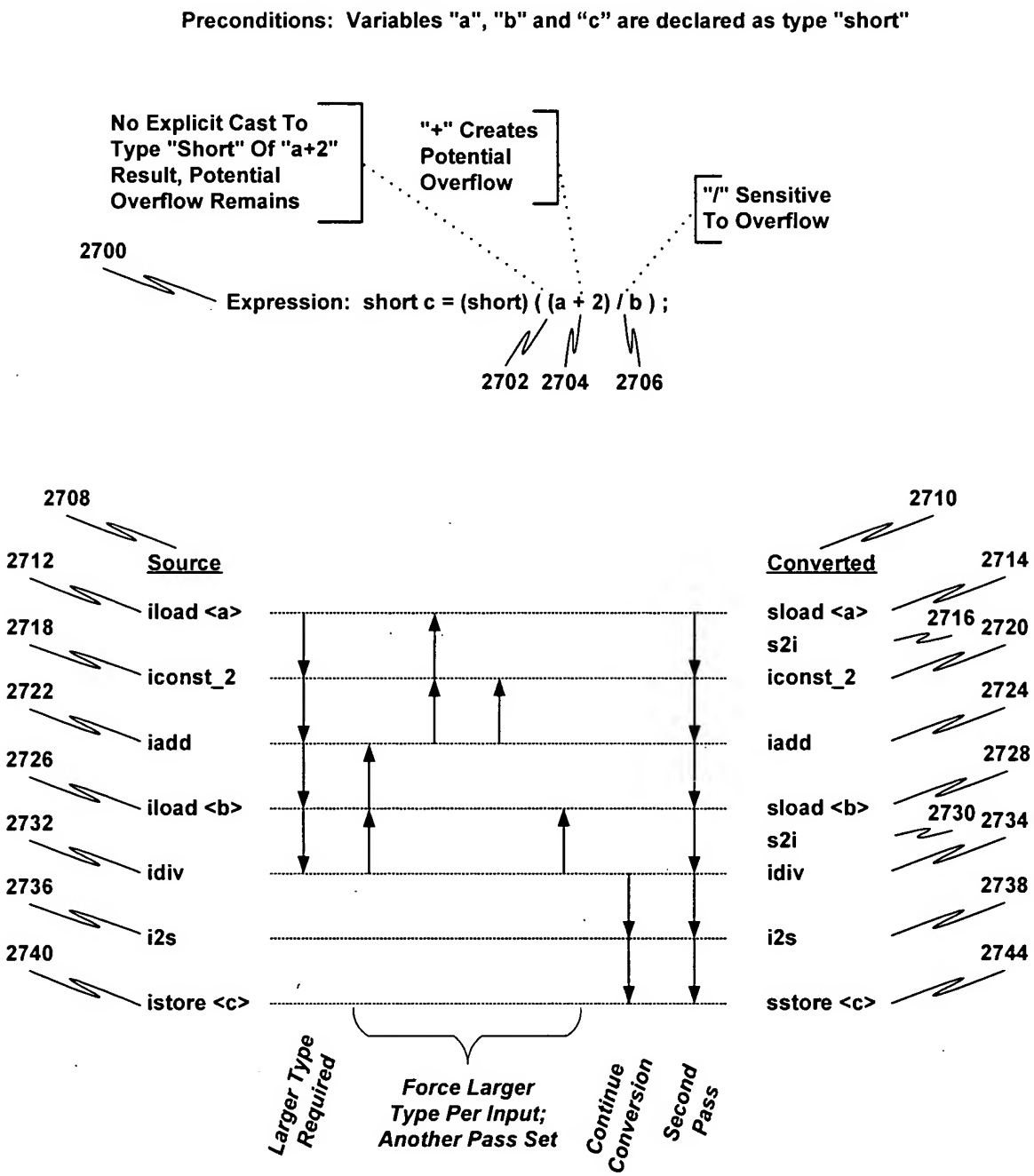

Preconditions: Variables "a", "b" and "c" are declared as type "short"
No Explicit Cast To Type "Short" Of "a+2" Result, Potential Overflow Remains
"+" Creates Potential Overflow
"/" Sensitive To Overflow
2700
Expression: short c = (short) ((a + 2) / b) ;
2702 2704 2706
2708
2710
2712
Source
Converted
2714
iload <a>
sload <a>
2716
2718
s2i
2720
iconst_2
iconst_2
2722
2724
iadd
iadd
2726
2728
iload <b>
sload <b>
2730
2732
s2i
2734
idiv
idiv
2736
2738
i2s
i2s
2740
2744
istore <c>
sstore <c>
Larger Type Required
Force Larger Type Per Input; Another Pass Set
Continue Conversion
Second Pass

FIG. 27

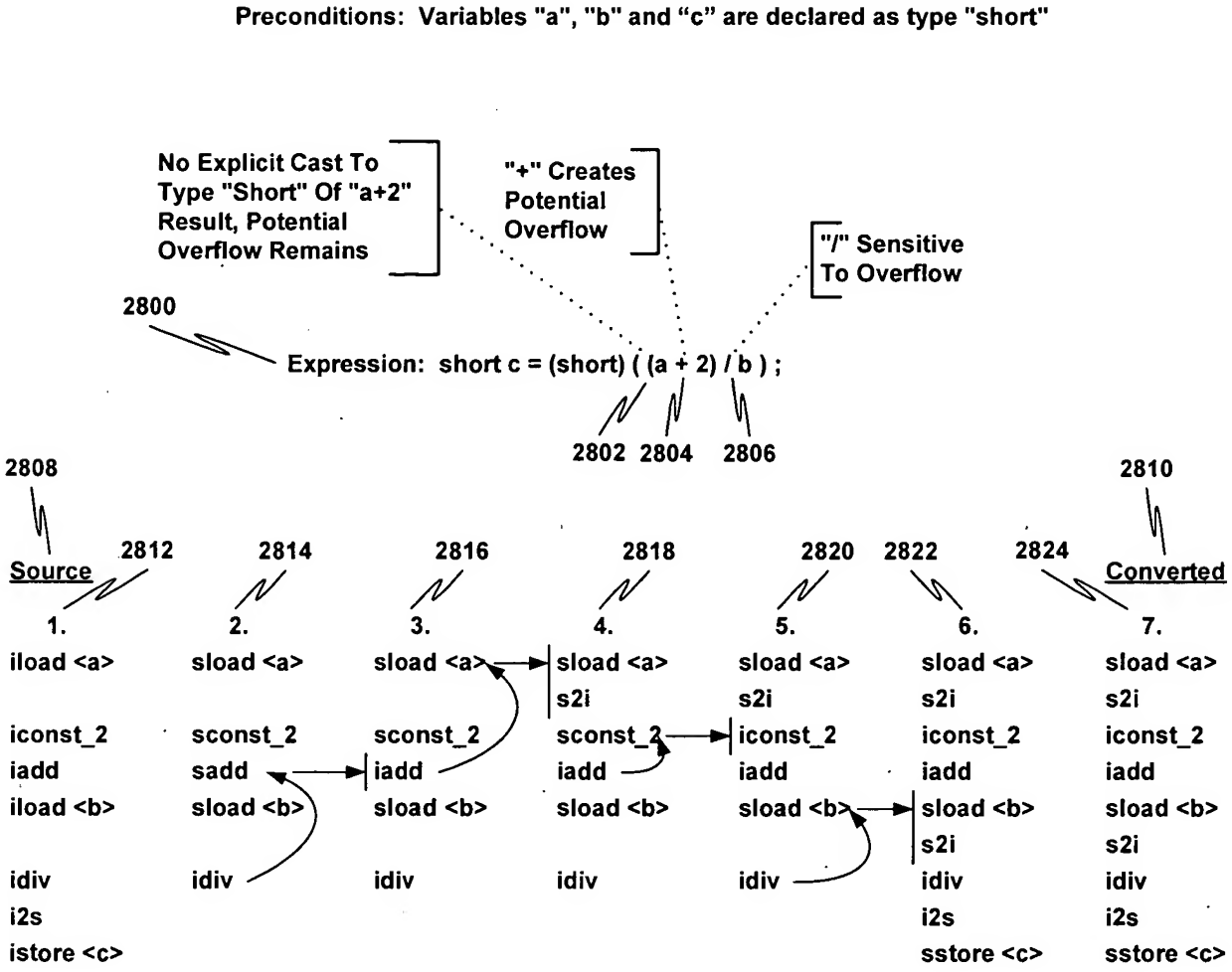

Preconditions: Variables "a", "b" and "c" are declared as type "short"
No Explicit Cast To Type "Short" Of "a+2" Result, Potential Overflow Remains
"+" Creates Potential Overflow
"/" Sensitive To Overflow
2800
Expression: short c = (short) ((a + 2) / b) ;
2802 2804 2806
2808
2810
Source
Converted
2812 2814 2816 2818 2820 2822 2824
1. 2. 3. 4. 5. 6. 7.
iload <a> sload <a> sload <a> sload <a> sload <a> sload <a> sload <a>
s2i s2i s2i s2i
iconst_2 sconst_2 sconst_2 sconst_2 iconst_2 iconst_2 iconst_2
iadd sadd iadd iadd iadd iadd iadd
iload <b> sload <b> sload <b> sload <b> sload <b> sload <b> sload <b>
s2i s2i s2i
idiv idiv idiv idiv idiv idiv idiv
i2s i2s i2s
istore <c> sstore <c> sstore <c>

FIG. 28

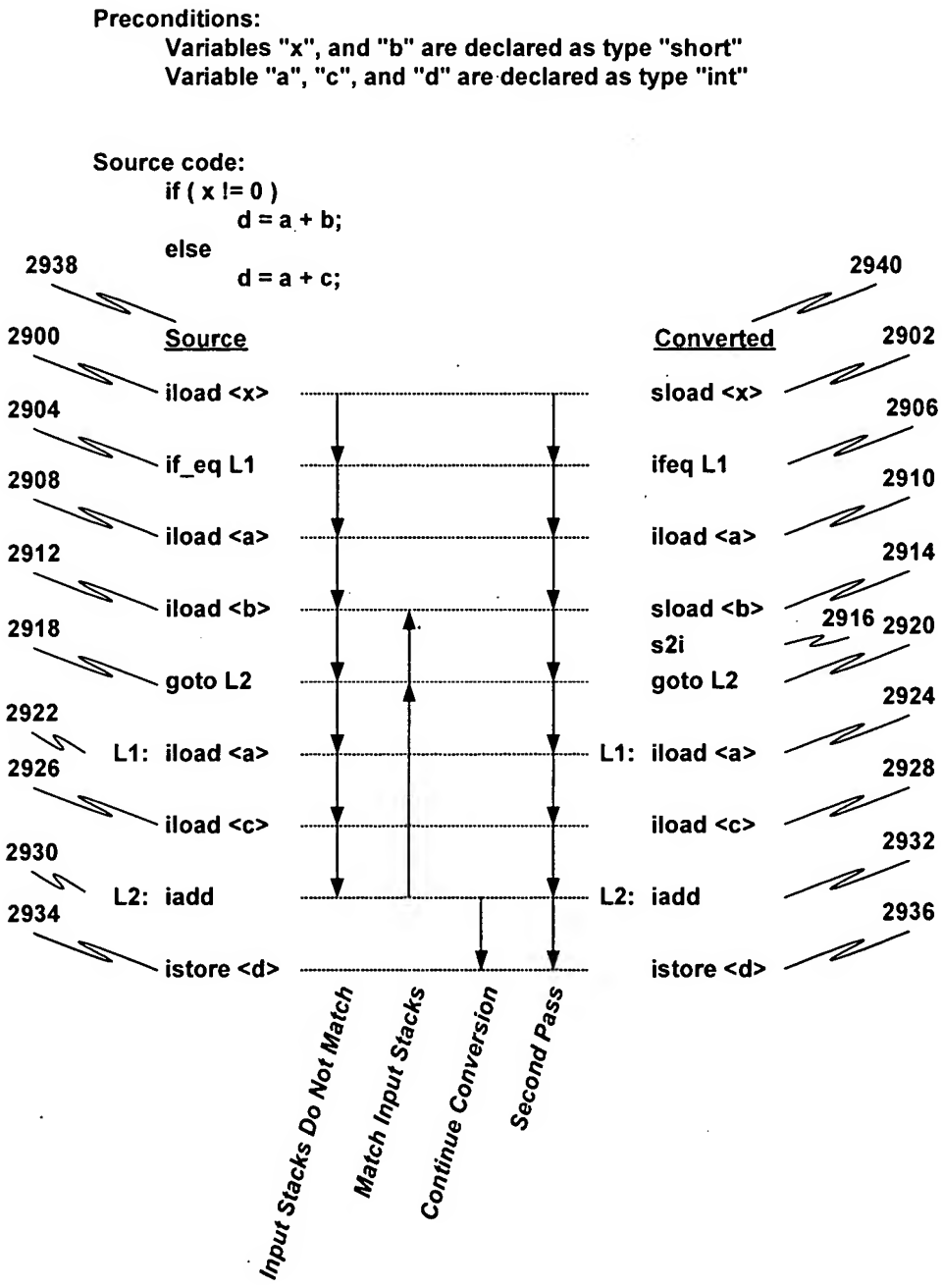

Preconditions:
Variables "x", and "b" are declared as type "short"
Variable "a", "c", and "d" are declared as type "int"
Source code:
if (x != 0)
d = a + b;
else
d = a + c;
2938
2940
Source
Converted
2900
2902
iload <x>
sload <x>
2904
2906
if_eq L1
ifeq L1
2908
2910
iload <a>
iload <a>
2912
2914
iload <b>
sload <b>
2916
s2i
2918
2920
goto L2
goto L2
2922
2924
L1: iload <a>
L1: iload <a>
2926
2928
iload <c>
iload <c>
2930
2932
L2: iadd
L2: iadd
2934
2936
istore <d>
istore <d>
Input Stacks Do Not Match
Match Input Stacks
Continue Conversion
Second Pass

FIG. 29

Preconditions:

Variables "x", and "b" are declared as type "short"
Variable "a", "c", and "d" are declared as type "int"

Source code:

```
if ( x != 0 )
    d = a + b;
else
    d = a + c;
```

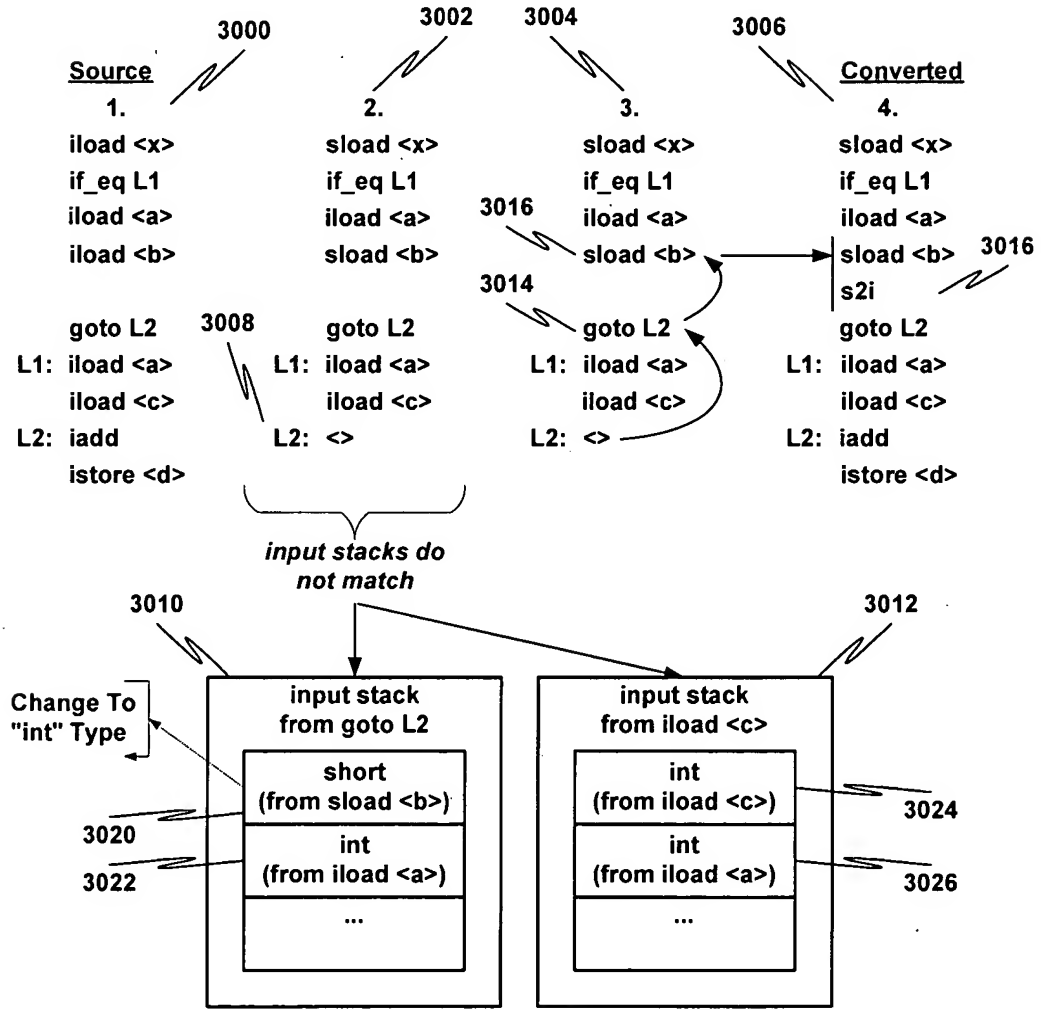


FIG. 30

Preconditions:

Variables "a", "b" and "c" are declared as type "short"
Variable "d" is declared as type "int'

Source Code:

```
c = (short) (a * b);
d = b + d;
```

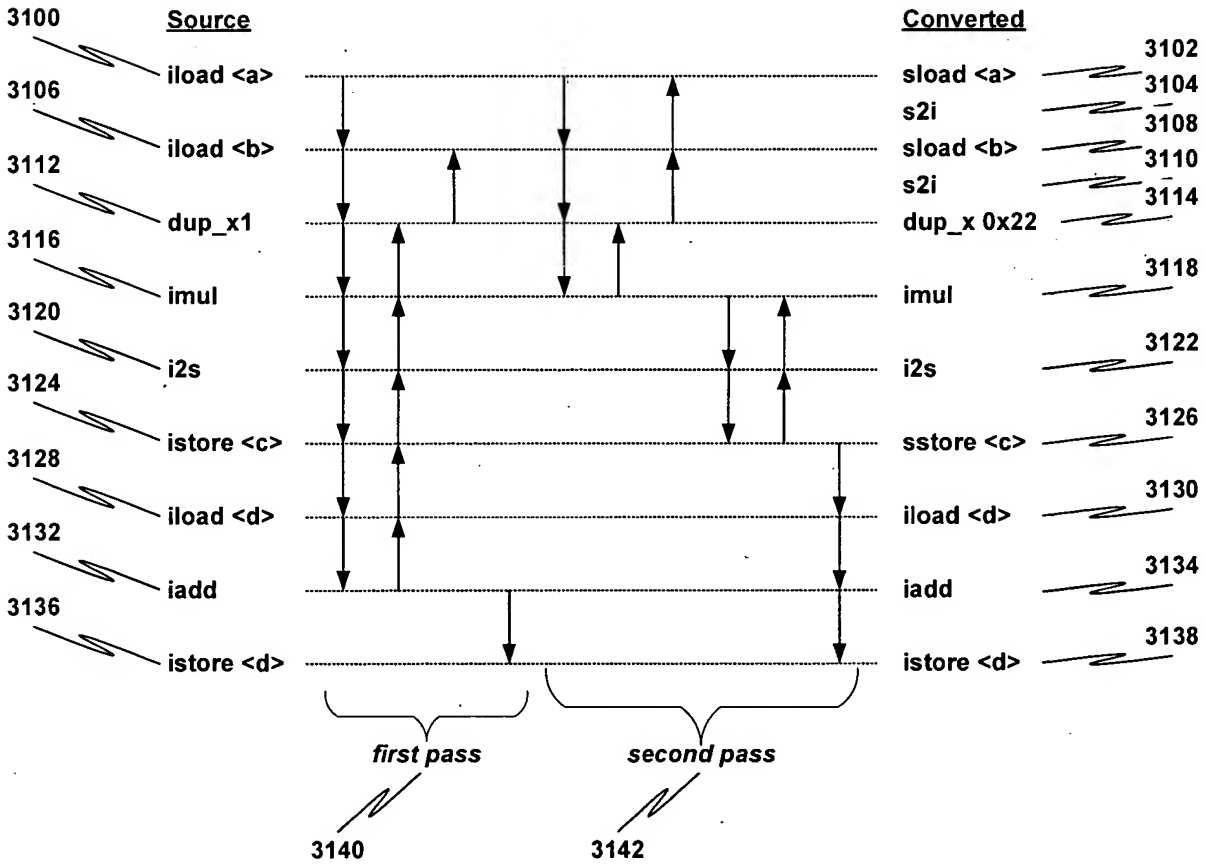


FIG. 31

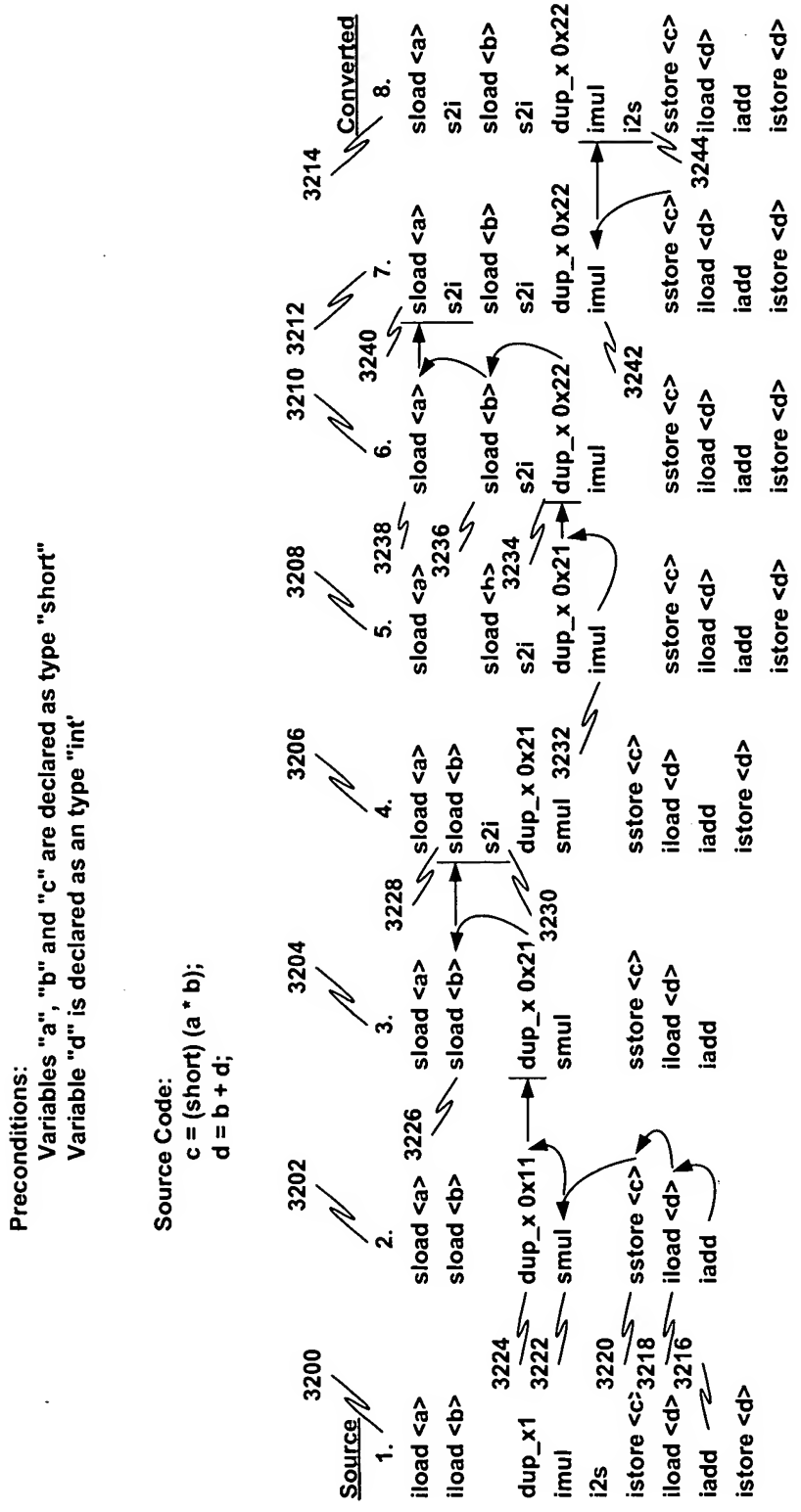

Preconditions:
Variables "a", "b" and "c" are declared as type "short"
Variable "d" is declared as an type "int"
Source Code:
c = (short) (a * b);
d = b + d;
Source
Converted
3200
3202
3204
3206
3208
3210
3212
3214
3216
3218
3220
3222
3224
3226
3228
3230
3232
3234
3236
3238
3240
3242
3244
1. iload <a>
iload <b>
2. sload <a>
sload <b>
3. sload <a>
sload <b>
4. sload <a>
sload <b>
5. sload <a>
sload <h>
6. sload <a>
sload <b>
7. sload <a>
sload <b>
8. sload <a>
sload <b>
dup_x1
imul
dup_x 0x11
smul
dup_x 0x21
smul
s2i
dup_x 0x21
imul
s2i
s2i
dup_x 0x22
imul
s2i
s2i
dup_x 0x22
imul
i2s
istore <c>
iload <d>
iadd
istore <d>
sstore <c>
iload <d>
iadd
istore <d>
sstore <c>
iload <d>
iadd
istore <d>
sstore <c>
iload <d>
iadd
istore <d>
sstore <c>
iload <d>
iadd
istore <d>

FIG. 32